Activity 8

Coding Exercise, CS 335

In this exercise we practice making a text file representation of a Turing machine and then “run” that TM on different inputs using the provided simulateTM Python program.

1. Download the WCBC source code which is provided to accompany our textbook. Click the “Python Programs” link on this website: https://press.princeton.edu/titles/11348.html

2. Unzip the downloaded zip file to a directory where you will put all your source code for CS 335.

3. Start a Python shell running on the directory where you saved the WCBC code files in step 2.

4. Import a couple of functions we will need, rf and simulateTM, then try using those functions:

   ```
   >>> from utils import rf
   >>> from simulateTM import simulateTM
   >>> simulateTM(rf('containsGAGA.tm'), '928GAGAxx987')
         (Should return: 'yes')
   >>> simulateTM(rf('containsGAGA.tm'), '928xyz')
         (Should return: 'no')
   ```

5. Open containsGAGA.tm in Sublime Text. Save as contains111.tm.

6. With paper and pencil, or at the whiteboard, plan out the state diagram for a TM to decide the Contains111 problem: For any string I which has “111” as a substring, F(I) = “yes”; for all other strings I, F(I) = “no”.

   - Recall that a “decider” cannot go into an infinite loop. For any input string over the corresponding alphabet, your TM must eventually halt in either the “accept” state or the “reject” state.
   - As a convenience, explicit transitions to the reject state are not required. If at any point in the computation there is not a transition provided for the current state and symbol, we will assume that the TM transitions to “qR”, the reject state, thus stopping the computation with a “no” answer.

7. Revise the contents of contains111.tm to create a text description of the Turing machine you designed for this problem.

8. Back in the Python shell, run simulateTM on ‘contains111.tm’ with the following inputs.

   - Negative examples: ""“,”1“,”x“,”11“,”xx“,”11x1“,”abcd11"
   - Positive examples: “111”, “1110”, “0111”, “xy111z”, “1111”

9. Download the file test_contains111.py[1]

   Copy the following code and save it as test_contains111.py.

```
##############################
# test_contains111.py        #
# Barb Wahl, 9–25–2018       #
##############################

from utils import rf
from simulateTM import simulateTM

def main():

   should_reject = ["", "1", "x", "11", "xx", "11x1", "abcd11"]
   should_accept = ["111", "1110", "0111", "xy111z", "1111"]
   run_tests("contains111.tm", should_reject, "no")
   run_tests("contains111.tm", should_accept, "yes")

def run_tests(prog_file, L, expected):
   for I in L:
      result = simulateTM(rf(prog_file), I)
      if result == expected:
         decorator = " — correct."
      else:
         decorator = " — ERROR!"
      print_report("contains111", I, result + decorator)

def print_report(fun_name, I, msg):
   print(fun_name + "(" + I + ") = " + msg)

main()
```

10. In the terminal, run test_contains111.py and verify that your contains111.tm description is passing the provided tests (fix any errors you find in your description).

11. Read through the code for test_contains111.py. You should be able to modify this code to test other deciding TM descriptions.

[1] https://skiadas.github.io/TheoryCompCourse/site/activities/test_contains111.py